Programming Structures and Abstractions (CS 2334)
Project 2

October 8, 2009

Introduction

In this project, you will be practicing your skills in modifying and defining classes and class hierarchies. Specifically, you will be adding more functionality to your Finch action representation that will allow behavior that is conditioned on the sensor states.

Milestones

1. Add an integer **priority** to all FinchActions (5 pts)
2. Create two new subclasses of FinchAction: **FinchGuardedJog** and **FinchGuardedOrientation** (20 pts)
3. Configure FinchAction to implement **Comparable** (15 pts)
4. Add a **sort** method to FinchActionList (15 pts)
5. Add a new **FinchActionList constructor** that creates a list from an existing list (15t pts)

Other components:

- Develop and use a proper design (UML and class stubs) (15 pts)
- Use proper documentation and formatting (javadoc and in-line documentation) (15 pts)

Note: of these last two components, a total of 20 points are available during the design phase (10 points for each). The remaining 80 points are obtainable for the final submission of the project.

This lab is due in two phases:

1. Thursday, October 1^{st} at 5:00pm: design.
2. Thursday, October 8^{th} at 5:00pm: completed program and short demonstration.

More details for what to hand-in and when may be found below.

Resources

Main class web page:

- Java JDK 6 Classes
- Finch Introduction
- Finch API: how to talk to your Finch
- FinchSoftwarev3_1.zip: core Finch code and example programs (download and install on your hard disk)

Main web page / projects / general :

- Finch Driver Setup: setting up your computer to talk to the Finch
- Finch Manual: general information about the Finch hardware
- Documentation_Requirements
- Submission Instructions

Main web page / projects / project2 :

- project2.pdf: this project description
- cs2334_project2.pdf: a copy of the lab section slides
- moves.txt: an example input file
- ... other example files to come...

Input Files

The input file specification has changed. In particular, every action now has an integer **priority** in addition to a name. In addition, we have introduced two new actions.

- Jog the finch:

  `<name> <priority> JOG <duration> <left> <right>`

- Change the beak color and wait for duration:

  `<name> <priority> RGB <duration> <red> <green> <blue> <darken>`

  Color channels are integers in the range of [0 ... 255]

  darken is either 0 or 1 and indicates whether the beak is turned off after the action is complete.

- Generate a sound of a given duration:

  `<name> <priority> SOUND <duration> <frequency>`

- Guarded jog: move the wheels of the finch until an obstacle is observed:

  `<name> <priority> GJOG <left> <right>`

- Guarded orientation: wait until the finch is in one of 4 different orientations

  `<name> <priority> ORIENT <orientation>`

  where **orientation** is one of the following strings: "up", "down", "upsidedown" or "level"

Example File

```
seek    19  JOG      1000    -10.0    -10.0
seek    10  GJOG     30.0    30.0
seek    38  GJOG     30.0    30.0
seek    27  JOG      2000    20.0     -20.0
seek    5   ORIENT   level
```

As before, after loading of this file into your data structure, the user will be able to search for a particular name and either display or execute the sequence of FinchActions that match the name. If the user specifies the name as "all", all of the FinchActions are displayed/executed in **name/priority order**.

Milestones

A milestone is a "significant point in development." Milestones serve to guide you in the design and development of your project. Listed below are a set of milestones for this project along with a brief description of each.

Milestone 1: Add priority to all FinchActions

Add a class variable called **priority** to all FinchActions. This should be a required parameter for the constructor (as is the name). Provide the appropriate accessor and mutator methods.

Note: this change should be made in conjunction with Milestone 2.

Milestone 2: Create two new FinchAction Child Classes

The two new action classes are **FinchGuardedJog** and **FinchGuardedOrient**.

FinchGuardedJog is similar to class FinchJog in that the velocities of the two wheels are specified. However instead of spinning the wheels for a specified amount of time, the wheels continue to spin until an obstacle is detected by either the left or right obstacle sensors. If an obstacle is detected immediately, then the wheels should not spin.

FinchGuardedOrientation is a class that waits for the Finch to be in one of four different orientations: up, down, upsidedown or level. If "upsidedown" is specified, then the execute method will wait until the Finch is placed on its back before returning.

For our purposes, the orientation may be stored as a String. If you take this approach, you should ensure that only the 4 valid strings are stored. Alternatively, you could create your own enum class.

See the Finch API documentation for a description of the methods that will tell you when your Finch is in a particular orientation.

Note: neither of these classes requires a duration class variable. You must reorganize your class structure so that the proper commonalities are reflected in your class hierarchy.

Milestone 3: FinchAction implements Comparable

The **FinchAction** class should be reconfigured to implement the **Comparable** interface. This includes a concrete implementation of the **compareTo()** method in the FinchAction class:

```
public int compareTo() { ...}
```

This method orders actions first by **name** (ignoring case) and then by **priority**.

Note: you must specify the Comparable implementation such that only FinchActions are compared with other FinchActions.

Milestone 4: Implement a sort method for FinchActionList

Implement a **sort()** method for the FinchActionList class:

```
public void sort() { ...}
```

In order to implement this method, use the implementation of sort() provided in the book (listing 11.10) with one modification: use the generic form of the method declaration so that it does not accept an array of **any** comparable objects, but instead accepts a specific class of Comparable objects.

The sort() method must be called by your driver class as soon as the file is loaded.

Milestone 5: Add a new FinchActionList constructor

This constructor takes as an input parameter an existing list and a string:

```
public FinchActionList(FinchActionList list, String key){ ... }
```

The returned list contains only those actions that match the specified string. Note that you will also need to provide a **getAction** method:

```
public FinchAction getAction(int i) { ... }
```

that returns the i^{th} action of the FinchActionList.

Use this new constructor to re-implement the **display()** and **execute()** methods (your implementations will be cleaner now).

Hand-In Procedures

Part 1: Design

Deadline: Thursday, October 1st at 5:00pm

Each group must hand in one copy of each of the following:

1. A printed cover page that lists the group members, work contributed by each, and any outside citations. Turn in the hardcopy to the TA or the lecturer.

2. UML diagram on engineering paper: turn in the hardcopy to the TA or the lecturer.

3. project2_design.zip to D2L. This is the zip file produced by Eclipse that contains:

   - Each of the classes with class variables, method header documentation and method stubs (prototypes). **Other than dummy return calls or calls to this() or super(), do not include any other code (points will be subtracted if other code is included).** These calls will enable you to compile your java code.
   - html description of your project produced by javadoc. Make sure to check that the resulting html files contain all of the correct information.

Part 2: Complete program and short demonstration.

Deadline: Thursday, October 8th at 5:00pm

Each group must do the following:

1. Turn in a printed cover page that lists the group members, work contributed by each, and any outside citations. Turn in the hardcopy to the TA or the lecturer.

2. Hand in the modified UML diagram on engineering paper: turn in the hardcopy to the TA or the lecturer.

3. Turn in project2.zip to D2L. This is the zip file produced by Eclipse that contains:

   - Each of the class implementations with documentation. The author(s) of each class should be documented at the top of the java file.
   - html description of your project produced by javadoc.

4. A short demonstration. We will reserve time during your laboratory section for you to demonstrate your working program. You may also attend office hours or make appointments to perform the demonstrations.

Hints

- Your program must be your own work. Do not discuss or look at the solutions of other groups in the class. However, you may discuss general issues (i.e., not directly related to the project requirements) with your classmates, as well as use the book and the resources available on the net.
- Start your work early. This is not a trivial programming assignment.
- Ask for help early. If you are stuck on something, talk to the TA or the instructor sooner than later (this is what we are here for).
- See the Finch API documentation for a list of methods that will allow you to access the sensors and to produce behavior.